On Imperfection in Information as an "Early" Crosscutting Concern and its Mapping to Aspect–Oriented Design

Miguel–Ángel Sicilia, Elena García
Computer Science Department, University of Alcalá
Polytechnic School, Ctra. Barcelona km 33.6
28871 – Alcalá de Henares (Madrid), Spain.
{*msicilia,elena.garciab*}*@uah.es*
http://www.cc.uah.es/msicilia

Abstract

Imperfection in information can be considered a cross–cutting concern that encompasses diverse kinds of imprecision, uncertainty or inconsistency management inside a software system. An early consideration of the type(s) and required level of imperfection handling for the system is necessary to properly inform design, and to serve as criteria for choosing the appropriate mathematical model(s) that will be implemented in the final software. In this paper, imperfection is discussed as an integral part of a concern–oriented approach to requirements and domain engineering, and some illustrative examples of the mapping of these concerns in aspect–oriented design are also provided.

1. Introduction

The *separation of concerns* principle has recently been applied to early stages of development like requirements engineering [15, 7, 3] and software architecture [16]. The early separation of cross–cutting concerns results in improved localization, and eventually in improved development and maintenance activities. In the research literature, a number of examples of cross–cutting concerns are often used for illustration purposes, or they are described when reporting case studies. Recurring examples include, for example, security, usability, persistence or performance.

The focus of this paper is that of *imperfection* in information as a cross–cutting system concern that is currently overlooked in many application models. Imperfection is a multifaceted concept including imprecision, uncertainty and inconsistency, being classical probability a model for a specific type of imperfection among many others. Currently, general and mature mathematical frameworks for the management of imperfection are available [5], although their widespread use in mainstream development technologies and industrial systems is still to come. To adhere to an unambiguous interpretation of the terms used for the various sub–aspects of information imperfection, we'll use them in the sense given in Smet's taxonomy described in [14]. It should be noted that this taxonomy reflects the diverse concepts of imperfection in information, and not their mathematical handling, so that it is related to *early* domain modelling, that will be *later* mapped to a concrete representation.

Imperfection in information should be addressed early in the lifecycle due to the specifics of uncertainty and imperfection in conceptual modeling [2], and its impact on architectural and implementation decisions, most notably in persistence and querying [10]. In any system dealing with imperfection, a mathematical model (or several of them) for its representation must be selected, according to the concrete concerns stated in the requirements and domain models. To do so, enough detail must be provided so that system tests could eventually be derived from them, in order to assess the validity of the representation chosen with regards to the required capabilities of the system.

In this paper we approach the problem of specifying information imperfection in software requirements and domain models as cross–cutting concerns, and some examples of mapping such concerns into aspects at the design and implementation stages. The motivation of our present work is that of initiating research in Software Engineering models that are conceptually connected with the diverse mathematical representations of imprecision and uncertainty, helping practitioners to make decisions regarding the use of the growing research results in the area. It should be noted that our focus is modeling imperfection that will be managed in the final system. Previous related work, e.g. [6] has focused on imperfection in the modeling process itself.

The rest of this paper is structured as follows. Section

2 examines how imperfection can be specified as an integral part of concern–oriented requirements and domain engineering activities, to a level of detail sufficient to inform subsequent architectural and modularization decisions. In Section 3, illustrative examples are given about the mapping of domain–level concerns to implementation structures. Finally, conclusions and a future outlook are provided in Section 4.

2. Imperfection as a Concern in Requirements and Domain Engineering

Information imperfection usually arises at the early stages of the development, since it pervades the description of the domain or real world situation. In fact, classical error theory that manifests in required degrees of precision in numerical computations is just a model to deal with a facet of imperfection related to numerical representation and error in measurement instruments. Probabilistic or statistical considerations arising, for example, in regression models or questionnaire–based tests are just a manifestation of uncertainty. In addition, some classes of systems work in a context that inherently entails imprecision and uncertainty, like personalized Web systems that use user's navigation to tailor their structure [12].

Information imperfection is a logical "matter of interest", according to COSMOS [15] terminology [1]. It can be organized in several *classifications*, one for each of the principal aspects of imperfection: `Imprecision-Related`, `Uncertainty-Related`, `Inconsistency-Related` or `Hybrid`. Classes inside those categories may refer to more specific types of imperfection according to Smet's taxonomy, e.g. `FuzzyElement` refers to "imprecision without error" without decidability as in "age is close to 30", while `PossibleElement` refers to "happen–ability" as a kind of uncertainty. In addition, imperfection manifestations can be classified in `Domain-Imperfection`, `UserImperfection` and `System-Imperfection` according to the source from which the imperfection originates. For example, inferences internal to the system may generate imperfect information from perfect inputs. According to the kind of element in the conceptual model that is subject to imperfection, we can have additional classifications, namely `ImperfectElement` and `ImperfectRelationship`, the former containing classes `ImperfectClass`, `ImperfectAttribute`, `ImperfectFunction` and `ImperfectResult` which roughly correspond to classes, attributes, method (or more generally, functionality), and method results in conceptual models, and the latter containing classes for each type of relationship (association, generalization, etc.). Imperfect conceptual model elements can be expressed in the domain model through extensions to the `UML` like the one sketched in [9]. Figure 1 depicts some of the just described elements and some of example relationships between them, representing classifications as `UML` packages.

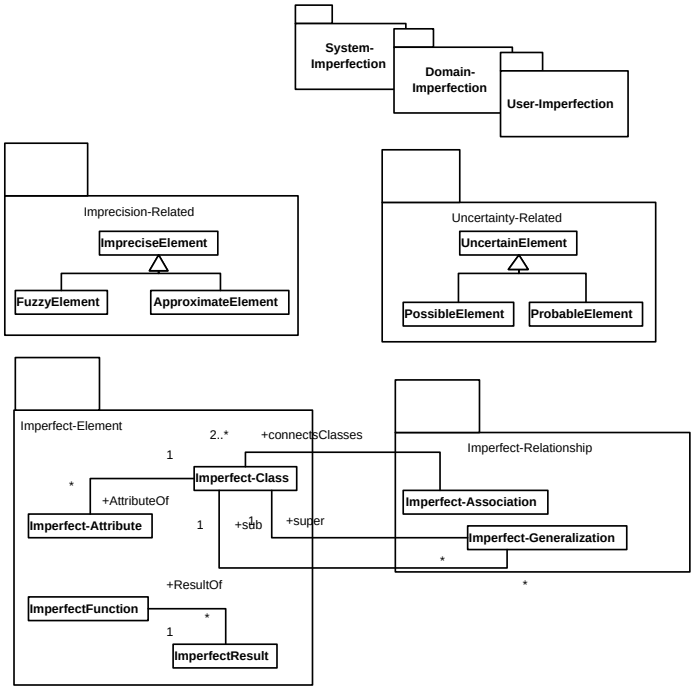

Figure 1. General concern dimensions related to information imperfection

Properties as defined in COSMOS are "concerns that characterize other logical concerns". Concerns that arise in the context of imperfection include `Granulation-level` and `Interpretability`. The former refers to the degree of "summarization" of an element, e.g. "large pages" may be interpreted to subsume "very–large" and "moderately large", thus summarizing information. The latter refers to the ease of interpretation of the information by humans, and is often considered as a quality criteria in rule bases. They can be used as requirements constraining the design of other concerns, e.g. `High-Interpretability` may involve selecting a fuzzy rule simplification algorithm at later stages for the computation of a given conceptual model element.

It should be noted that the imperfection–related concepts introduced so far do not require understanding neither about the mathematical frameworks for uncertainty handling not about their software representation, so that they can be con-

[1] In what follows, concern–modeling concepts are taken from COSMOS, in spite of the fact that the matter dealt with here is related to a greater extent to domain modeling, while COSMOS is intended for earlier stages of development.

sidered a concern in domain and requirements engineering.

In what follows, we briefly sketch concern space analysis with regards to imperfection for two case studies.

2.1. Case Study: Market Segmentation

Market segmentation systems provide support for the classification of customers with the purpose of targeting marketing strategies [17]. Market segmentation criteria are in many cases uncertain, and the resulting subsets can be considered to have no sharp boundaries, specially in the new relationship marketing paradigm [13].

A basic model for market segmentation using the relationship value model in [13] will result in the classes and instances showed in Table 1.

Table 1. Example imperfection–related concerns in a basic market segmentation setting.

Instances	Classes/classifications
Expected-Incremental-Purchases	SubjectiveUncertainElement, User-Imperfection, ImperfectAttribute
Estimated-Relationship-Duration	SubjectiveUncertainElement, User-Imperfection, ImperfectAttribute
Relationship-Value	ImpreciseElement, System-Imperfection, ImperfectResult
Customer-Segment	FuzzyElement, System-Imperfection, ImperfectClass
Customer-Similarity	FuzzyElement, System-Imperfection, ImperfectAssociation
Net-Relationship-Value-Model	FuzzyElement, System-Imperfection, ImperfectFunction

Segments can be considered imperfect classes of users inferred by the system from a net value relationship model that estimates the value of each customer relationship from rough estimates of increments in purchases and expected relationship duration. Those estimates are rough pessimistic and optimistic values obtained from experts. Customer similarity is derived from the segments and perhaps from the analogies in purchasing behavior of a pair of customers, as often interpreted in recommender systems [8].

2.2. Case Study: Adaptive Learning System

Adaptive Learning Systems are characterized by tailoring the hypermedia structure of the learning contents to the knowledge and/or characteristics of the learners. The case study described here is based in the adaptive technology described in [11] where a rule–based system was used to adapt the presentation of links ("fuzzy links") in the courseware. The main imperfection–related concerns are showed in Table 2.

Table 2. Example imperfection–related concerns in a rule–based adaptive Web system.

Instances	Classes/classifications
Knowledge-about-Lesson$_i$	SubjectivePossibleElement, System-imperfection, Imperfect-Association-Attribute
Fuzzy link	FuzzyElement, Domain-Imperfection, Imperfect-Class
Rule$_1$, ... Rule$_n$	FuzzyElement, User-Imperfection, Imperfect-Function
User-Categorization-Inference	UncertainElement, FuzzyElement, System-Imperfection, Imperfect–Class
Content-Tailoring-Inference	FuzzyElement, System-Imperfection, Imperfect–Attribute
Content-Tailoring-Algorithm	FuzzyElement, System-Imperfection, Imperfect–Function
Learner-Style	ObjectivePossibleElement, FuzzyElement, Imperfect-Class, Uncertain-Class

The knowledge a given learner is supposed to have about a given lesson can be modeled as an association attribute, a specialization of `Imperfect-Association`. The kind of uncertainty for such knowledge levels is considered in terms of epistemic possibility. Fuzzy links represent typed semantic relationships about nodes or contents, so that the vagueness expressed in them come from the domain being teach. Rules use fuzzy links and other model characteristics both to infer imprecise categories of users and also to establish personalized content attributes for each user or group. Rules are elicited from experts, so that it can be considered that the source of imperfection are a special kind of users. Content tailoring algorithms are an alternative to rules in the form of predefined function that obtains imprecise personalization of content attributes. Learner styles are both uncertain and imprecise, since the techniques used to obtain them, although based on recurring patterns of interaction, do not provide reliable answers.

3. Mapping Concerns of Fuzziness to Aspect–Oriented Design

In this section, a number of concrete mappings of information imperfection concerns are described for the purpose of illustration.

3.1. Fuzzy arithmetics

Numerical `ImperfectAttribute`s can be mapped to fuzzy numbers and fuzzy arithmetics [4], irrespective of the kind of imperfection (imprecision, uncertainty or both) they represent. This is the case of the uncertain `Expected-Incremental-Purchases` and `Estimated-Relationship-Duration` values that are used to compute the `Net-Relationship-Value-Model` in the market segmentation case study. The design of systems dealing with fuzzy numbers requires a library supporting them like the `Fuzzy Java Toolkit`. Nonetheless, it would be desirable that fuzzy arithmetics become an standard part of programming language libraries, so that developers could select using crisp or fuzzy arithmetics. The arithmetics of fuzziness can be modularized in aspects that modify the behaviors of classes representing numbers. For example, let's consider an hypothetical extension to Java `Double` class providing arithmetic functions [2] with an interface like the following:

```
public class ArithmeticDouble extends Double{
   public add(Double x){ ... }
   public times(Double x){ ... }
   //...
}
```

Fuzzy arithmetic can be modularized in aspects by *introducing* attributes to describe the shape of the fuzzy numbers and intercepting calls to *getter* methods like `doubleValue()` to compute the appropriate defuzzification of the triangular number, as sketched in what follows:

```
public aspect FuzzyArithmetics{

  // Introduces upper and lower bounds
  // of triangular fuzzy numbers:
  private double ArithmeticDouble.upper;
  private double ArithmeticDouble.lower;


  double around  (ArithmeticDouble d):
        target(d) &&
        call(public double doubleValue()){
        double crisp = proceed(d);
        // defuzzify triangular number
        // if required...
        return crisp;
  }


}
```

This enables the co–existence of normal and fuzzy numbers in the same framework, as an option to changing programming language support by special–purpose interfaces.

3.2. Fuzziness in databases and queries

In almost every application, the use of specialized representations for imperfect information result in special persistence requirements. In consequence, database programming interfaces require extensions for such purpose. Concretely, here we focus in orthogonal persistence interfaces similar to those exposed by `Jdo`[3]. The extension for fuzziness of such kind of interfaces can be accomplished by adding elements to the query syntax — as is done in [1]— and also by augmenting programming interfaces to deal with the desired fuzzy modeling capabilities — e.g. as in [10]. Aspect–oriented design can be used to extend existing programming libraries for fuzziness without obscuring their original design. In what follows, we briefly sketch some design points of the extension of the `OJB`[4] interfaces for illustration purposes.

The core of such extensions is adding new schemata that van be made processed by the libraries by an aspect like the following:

```
public aspect FuzzyMetadataManagement
{
    private DescriptorRepository globalRep;
    void around (MetadataManager m):
         target(m) && call(
           * MetadataManager.init(..)){
        try{
                proceed(m);
        }catch(MetadataException e){throw e;}
        globalRep = loadFuzzyDesRep();
        m.mergeDescriptorRepository(globalRep);
    }
private DescriptorRepository loadFuzzyDesRep()
 {  //  load descriptor repository..  }
    //...
}
```

Then, the extended meta–schema describing the storage details of each kind of imperfect modeling concern becomes available. For example, explicit storage of membership values of objects belonging to fuzzy classes can be achieved through the following design, which introduces a new method in the `PersistenceBrokerImpl` class.

[2] this is actually not possible, since the class is declared `final`.

[3] http://access1.sun.com/jdo/

[4] http://db.apache.org/ojb/

```
public aspect FuzzyStorageHandling{
    public void PersistenceBrokerImpl.store(
     Object obj, Double m,  String fuzzyClass)
     throws PersistenceBrokerException {
        store(obj);
        storeMembership(obj, m, fuzzyClass);
    }
 //...
}
```

Such aspect code could be automatically generated from models with elements that have been specified with the `FuzzyElement` and `ImperfectClass` model concerns. In addition, the retrieval of fuzzy grades from the database can be accomplished by wrapping result collections with the following design:

```
public aspect FuzzyObjectWrapping{
    Collection around (PersistenceBroker p):
    target(p) && args(query)
    && call(Collection PersistenceBroker.
    getCollectionByQuery(Query query)){
           Collection aux=null;
             try{
                  aux =  proceed(p, query);
            }catch(PersistenceBrokerException e)
                {throw e;}
            return this.wrapResultCollection(aux);
    }
    // ...
}
```

Such kind of aspect–oriented design elements point out the possibility of a comprehensive model–based database implementation of the concerns of fuzziness through standard mappings.

4. Conclusions and Future Research Directions

Information imperfection can be considered a cross–cutting concern that arises at early stages of the development lifecycle. A concern–oriented requirement and domain analysis process can be used for the early specification of requirements for imperfection handling and their associated domain model elements, so that design and implementation decisions can be based on them. A tentative concern space analysis for information imperfection has been described, along with some examples of the mapping of concerns to specific aspect–oriented design options.

Further work is needed in the analysis of concerns regarding imperfection and its mapping to the range of available mathematical models that can be used to map them into design. Such analysis would eventually come up with the seamless integration of Fuzzy Set Theory and other related frameworks [5] into the software engineering process, facilitating the adoption and development of fuzzy techniques in all industrial areas. In addition, the UML language and its associated tools should be extended to explicitly address the requirements of requirements and domain engineering regarding imperfection (preliminary work is described in [9]).

References

[1] Callens, B. de Tré, G., Verstraete, J., Hallez, A.: A Flexible Querying Framework (FQF): Some Implementation Issues. Lecture Notes on Computer Science 2869: Proceedings of International Symposium of Computer and Information Sciences (ISCIS) 2003, 260–267.

[2] Chen, G. Fuzzy logic in data modeling : semantics, constraints, and database design, Kluwer Academic Publishers, 1998.

[3] Grundy, J. Aspect-Oriented Requirements Engineering for Component-based Software Systems. In Proceedings of the 4th IEEE International Symposium on Requirements Engineering. IEEE Computer Society Press (1999): 84–91.

[4] Kaufmann, A., and M. M. Gupta. Introduction to fuzzy arithmetic: theory and applications. New York: Van Nostrand Reinhold, 1985.

[5] Klir, G., Wierman, M.: Uncertainty-Based Information: Elements of Generalized Information Theory. Springer-Verlag (1998).

[6] Marcelloni, F. and Aksit, M. Fuzzy logic based object-oriented methods to reduce quantization error and contextual bias problems in software development. Fuzzy Sets and Systems (2004) (to appear).

[7] Rashid, A., Sawyer, P., Moreira, A. and Araujo, J. Early Aspects: A Model for Aspect-Oriented Requirements Engineering. In Proceedings of the IEEE Joint International Conference on Requirements Engineering. IEEE Computer Society Press. (2002): 199–202.

[8] Sarwar, B. M., Karypis, G., Konstan, J. A., and Riedl, J. "Analysis of Recommender Algorithms for E-Commerce". In proceedings of the ACM E-Commerce 2000 Conference. Oct. 17-20, 2000, pp. 158-167.

[9] Sicilia, M. A., García, E., Gutiérrez, J. A. (2002). Integrating fuzziness in object oriented modelling languages: towards a fuzzy-UML. In: Proceedings of the International Conference on Fuzzy Sets Theory and its Applications (FSTA 2002), 66-67.

[10] Sicilia, M.A., García, E., Díaz, P. and Aedo, I.: Extending Relational Data Access Programming Libraries

for Fuzziness: The fJDBC Framework. In: Proceedings of the 5th International Conference on Flexible Query Answering Systems. Lecture Notes in Computer Science 2522, Springer (2002):314–328

[11] Sicilia, M.A., García, E., Díaz, P., Aedo, I. (2002): LEARNING LINKS: Reusable Assets with Supports for Vagueness and Ontology-Based Typing. In *Proceedings of the International Conference on Computers in Education*, 1567-1568

[12] Sicilia, M.A. (2003). Observing web users: conjecturing and refutation on partial evidence. In *Proceedings of the 22nd North American Fuzzy Information Processing Society*, NAFIPS 2003, 530 -535.

[13] Sicilia, M.A., García, E. On Fuzziness in Relationship Value Segmentation: Applications to Personalized e-Commerce. ACM SIGECOM Newsletter, 4(2):1–10.

[14] Smets, P.: Imperfect information: Imprecision-Uncertainty. Uncertainty Management in Information Systems: From Needs to Solutions. Kluwer Academic Publishers (1997), 225-254.

[15] Sutton Jr., S.M. and Rouvellou, I. Modeling Software Concerns in Cosmos. In *Proceedings of the First International Conference on Aspect–Oriented Software Development* (AOSD 2002), Enschede, The Netherlands, Apr. 2002, ACM Press, 127-133.

[16] Tekinerdogan, B. ASAAM: Aspectual Software Architecture Analysis Method. In Proceedings of the Aspect–Oriented Requirements Engineering and Architecture Design Workshop, Boston, US, 2002.

[17] Wedel, M., Kamakura, W.A. 1999. Market Segmentation: Conceptual and Methodological Foundations, Kluwer Academic Publishers, 2nd edition.